Algorithmics Exam

Question 1: Find similar patterns of colors across multiple images

Paper: FINDING COLOR AND SHAPE PATTERNS IN IMAGES by Scott Cohen

Book: NI Vision Concepts Help

Article: **Exploring Image Similarity Approaches in Python (Medium) by Vasista Reddy, Search-by-Colour (Medium) by Joshua Comeau, The Microsoft Windows Palette Manager(compuphase)**

AI: ChatGbt (To fix the code bugs, to enhance the performance, understanding code logic)

Image similarity can be thought of as a numerical representation of how alike two images are in terms of their visual content. There are several dimensions along which images can be similar, such as color, shape, texture, and composition.

I have used the following methods to identify the similar patterns of colors across multiple images,

1. Histogram Based Approach
2. Structural Similarity Index (SSIM)
3. Color Feature Extraction - Dominant Colors

Histogram Based Approach:

The idea behind this approach is to get the histogram of the given images and compare it with each other to identify their similarities and errors. The method is based on fitting the histogram of the measured image to the histogram of a model function, and it can be used for contrast determination in fringe patterns. Simulated and experimental results are presented.

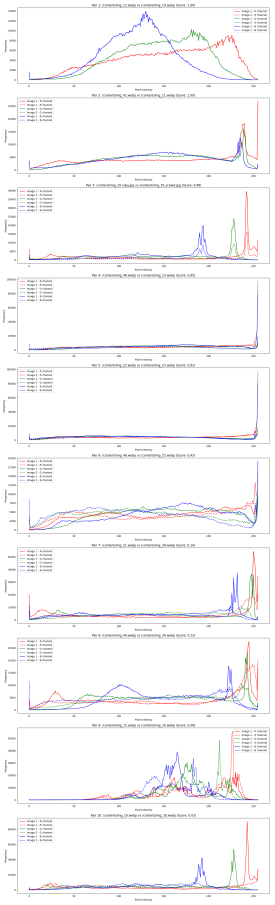
- A histogram represents the distribution of pixel intensity values for an image (e.g., for each color channel like red, green, blue).
- This method computes histograms for two images and compares them using metrics like correlation, intersection, or chi-square.
- It captures the overall distribution of colors in the images, ignoring spatial layout.

Strengths:

- **Robustness**: Works well for global comparisons; insensitive to minor spatial variations or transformations like rotation or translation.
- **Computational Simplicity**: Easy to implement and compute, making it suitable for large datasets.
- **Universal Application**: Does not require images to be the same size or resolution.

Weaknesses:

- **Spatial Insensitivity**: Ignores the spatial arrangement of colors. Two images with identical color distributions but different compositions can have identical histograms.
- **Lighting Sensitivity**: Results can vary significantly with changes in brightness or contrast.



Color Moments - Dominant Colors (K-Means Clustering)

- K-Means clustering groups similar pixels in an image based on their RGB values, identifying a set of dominant colors.
- The image is reduced to a compact representation based on these dominant colors.
- The similarity is computed using distances between the clusters of two images (e.g., Euclidean distance).

Strengths:

- **Compact Representation**: Summarizes the image in terms of its most dominant colors, reducing noise and irrelevant details.
- **Effective for Visual Themes**: Works well for images with distinctive color palettes, such as sunsets or forests.
- **Flexible**: The number of clusters (`k`) can be adjusted to control the level of abstraction.

Weaknesses:

- **Loss of Detail**: Fine-grained color variations or small features may be ignored, as the focus is on dominant colors.
- **Clustering Dependency**: The effectiveness depends heavily on the choice of `k` and the initial placement of cluster centroids.
- **Color Space Sensitivity**: Differences in color space representation (e.g., RGB vs. LAB) can affect clustering outcomes.

Output:

Top 10 Similar Images (Dominant Colors):

Structural Similarity Index (SSIM)

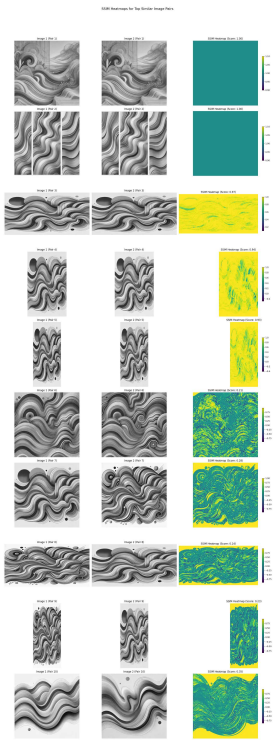
- SSIM evaluates how structurally similar two images are by comparing luminance, contrast, and spatial structure.
- It gives a similarity score between -1 and 1, where 1 indicates identical images.
- Unlike histograms or dominant color methods, SSIM focuses on perceptual quality, taking spatial relationships into account.

Strengths:

- **Perceptual Relevance**: Closely aligns with human visual perception, as it compares the structural content of images.
- **Localized Comparison**: Can highlight specific areas of similarity or dissimilarity, making it ideal for regions-of-interest comparisons.
- **Comprehensive**: Simultaneously considers multiple aspects (brightness, contrast, structure) rather than just color.

Weaknesses:

- **Same Dimension Requirement**: Both images must have identical dimensions, necessitating preprocessing like resizing or cropping.
- **Background Interference**: Differences in background content can heavily influence the similarity score.
- **Not Robust to Transformations**: Sensitive to geometric transformations (e.g., rotation, scaling).



a)Single Vertical Line

1. Cosine Similarity

- **Definition**:
  - Computes the cosine of the angle between two vectors.
  - Suitable for identifying the direction of patterns (e.g., RGB distributions) rather than their magnitude.
- **Advantages**:
  - Effective for comparing color patterns across vertical lines, even if their overall brightness differs.
  - Insensitive to uniform scaling of pixel values.
- **Limitations**:
  - Less effective if there are distortions or spatial misalignments.

2. Dynamic Time Warping (DTW)

- **Definition**:
  - A sequence alignment method that calculates the distance between two temporal sequences, accounting for distortions.
  - Matches sequences by stretching or compressing parts to minimize distance.
- **Advantages**:
  - Handles sequences with slight shifts, making it ideal for vertical lines with similar patterns but minor misalignments.
  - Robust to temporal distortions (e.g., intensity shifts along the column).
- **Limitations**:
  - Computationally intensive for large datasets.
  - Sensitive to noise in pixel values.

---

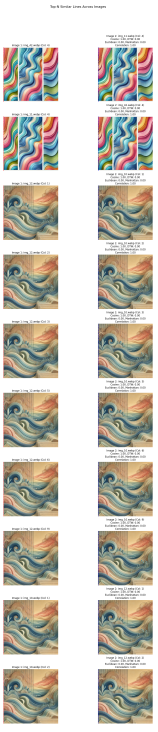
3. Euclidean and Manhattan Distance

- **Euclidean Distance**:
  - Measures the straight-line distance between two vectors.
  - Computes the square root of the sum of squared differences between corresponding elements.
- **Manhattan Distance**:
  - Measures the absolute distance between two vectors.
  - Computes the sum of absolute differences between corresponding elements.
- **Advantages**:
  - Simple and fast to compute.
  - Works well for straightforward comparisons.
- **Limitations**:
  - Sensitive to distortions or shifts in the sequences.
  - Ignores structural alignment and relationships.

---

4. Correlation

- **Definition**:
  - Measures the strength and direction of the linear relationship between two sequences (e.g., pixel intensities).
- **Advantages**:
  - Identifies proportional changes in pixel intensity between two vertical lines.
  - Effective when patterns are linearly related.
- **Limitations**:
  - Less robust for nonlinear relationships or distortions.



b)Band of lines:

- A band consists of several consecutive vertical lines grouped together (e.g., 2px, 5px, 10px, etc.).

- Each band is treated as a single entity for comparison.

Aggregation of Band Features:

- **Convert Bands into Single Representations**:
  - Aggregate the pixel values across the band (e.g., by averaging, summing, or concatenating values).
  - Each band becomes a 1D vector or a sequence of average RGB values.
- **Direct Distance Across Pixels**:
  - Treat the band as a multi-dimensional sequence and compute similarity across the entire band simultaneously.

Apply Similarity Measures:

- Similarity measures (cosine similarity, DTW, Euclidean distance, etc.) are extended to compare aggregated features or sequences of bands.

Experiment with Band Widths:

- Test different band widths (e.g., 2px, 5px, 10px, 20px) and compare results.
- Wider bands capture more global patterns, while narrower bands focus on finer details.

Band-Width Evaluation

1. Narrow Bands (2px, 5px)

- **Characteristics**:
  - Focus on fine-grained details.
  - More sensitive to local variations (e.g., texture changes, noise).
- **Advantages**:
  - Ideal for detecting subtle differences in images with fine patterns.
  - Captures local color and texture variations effectively.
- **Limitations**:
  - Sensitive to noise, making comparisons less robust for real-world images.

2. Moderate Bands (10px, 20px)

- **Characteristics**:
  - Balance between local detail and global context.
  - Capture meaningful patterns without being overly sensitive to minor distortions.
- **Advantages**:
  - Provide a good trade-off between accuracy and robustness.
  - Effective for most practical applications, such as finding similar textures or regions.
- **Limitations**:
  - May miss very small or subtle differences.

3. Wide Bands (50px and beyond)

- **Characteristics**:
  - Capture global patterns and general color distributions.
  - Less sensitive to local variations.
- **Advantages**:
  - Robust to noise and minor distortions.

- Suitable for detecting large-scale similarities, such as overall image themes.

- **Limitations**:
  - May overlook fine details or small differences between images.
  - Computation becomes heavier as the width increases.

Comparing Bands

1. Aggregating Band Features

- **Averaging**:
  - Compute the mean pixel value for each column within the band.
  - Simple and effective for reducing dimensionality.
- **Summing**:
  - Sum pixel intensities for each column.
  - Preserves overall intensity patterns but can distort relative relationships.
- **Concatenating**:
  - Treat all columns as a single sequence.
  - Preserves all details but increases computational complexity.

2. Simultaneous Distance Across Bands

- **Dynamic Time Warping (DTW)**:
  - Compare entire bands by aligning sequences along both the horizontal and vertical dimensions.
  - Effective for handling shifts or distortions across the band.
- **Cosine Similarity**:
  - Compute the angular similarity of the aggregated feature vectors.
  - Robust to intensity scaling, especially for color-based comparisons.
- **Euclidean and Manhattan Distances**:
  - Measure pixel-wise differences directly.
  - Simple but sensitive to distortions.

Output: Kindly refer the code, since the output visualization is huge, i didn’t include it here

Conclusion

- **Moderate Band Widths (10px to 20px)**:
  - Typically deliver the most credible results, balancing fine details and global patterns.
- **Wider Bands (>50px)**:
  - Better for general similarities but may overlook finer details.
- **Narrow Bands (2px to 5px)**:
  - Best for applications requiring high sensitivity to local changes but are more noise-prone.

C) Effects of Resizing the images

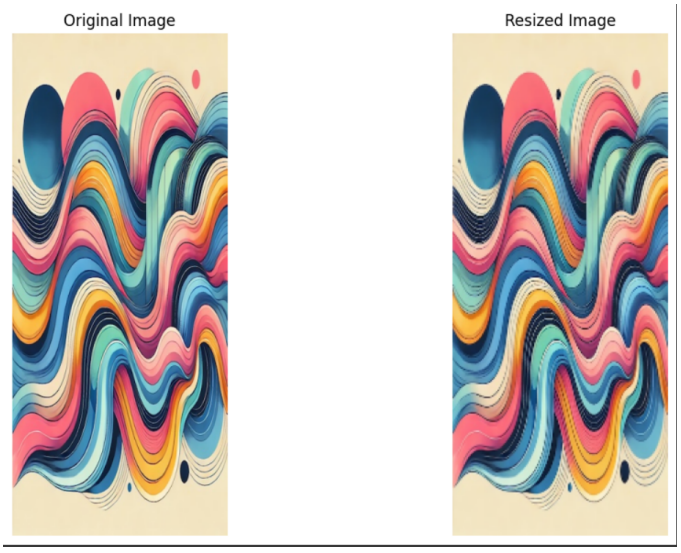
1. **Uniform Scaling**:
   - Scaling both width and height by the same factor (e.g., 50%, 200%).

- Maintains the aspect ratio of the image.

2. **Non-Uniform Scaling (Stretching)**:
   - Scaling width and height by different factors (e.g., width 50%, height 200%).
   - Changes the aspect ratio, introducing distortions.
3. **Extreme Resizing**:
   - Downscaling to very small dimensions (e.g., 10x10) or upscaling to very large dimensions.

Output: I have included only one scale due to space issue, kindly refer the code output

Testing for scale_x=0.5, scale_y=0.5



Question 3:

Report on the Photo Mosaic Art

1. Motivation

The choice of creating a **photo mosaic** is driven by its artistic appeal and computationally intriguing nature. Photo mosaics are a unique way of transforming an image into a collage of smaller images, preserving the essence of the original picture while adding layers of complexity and beauty.

2. Solution / Idea

The mosaic creation algorithm works by dividing the input image into a grid of tiles, calculating the average color for each tile, and replacing it with a smaller image (tile) from a precomputed image library that matches the color closely.

Steps:

1. Preprocess the image library by calculating the average color of each image and storing it in a cache for fast access.
2. Divide the input image into tiles of a specified size.
3. For each tile:
   - Calculate the average color.
   - Find the closest matching image from the library using Euclidean color distance.
   - Replace the tile with the resized matching image.

4. Save the final mosaic as an output image.

---

3. Pseudocode

```
Algorithm: Photo Mosaic Generation

Input: InputImage, ImageLibrary, TileHeight, TileWidth
Output: MosaicImage

1. Precompute average colors for all images in ImageLibrary
    For each image in ImageLibrary:
        Compute AverageColor
        Store (AverageColor, ImagePath) in Cache

2. Divide InputImage into tiles of size TileHeight x TileWidth

3. For each tile in InputImage:
    a. Compute AverageColor of the tile
    b. Find ClosestColor in Cache using Euclidean distance
    c. Replace tile with resized image corresponding to ClosestColor

4. Save the resulting MosaicImage
```

---

4. Computational Complexity

1. **Preprocessing**:
   - **Color Calculation**: , where is the number of library images, and are their average dimensions.$O(N \cdot H \cdot W)$
   - Total: , assuming images are small and average dimensions are constant.$O(N)$
2. **Mosaic Generation**:
   - **Tile Matching**: , where is the number of tiles in the input image, and is the number of images in the library.
   - Total: $O(T \cdot M)$
3. **Overall Complexity**: $O(N + T \cdot M)$.

Compute Time Implications:

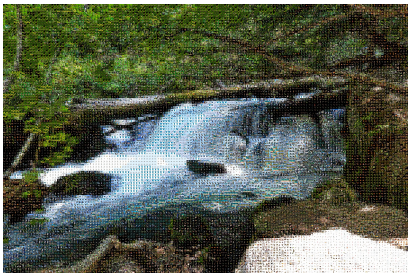
- For a 500x500 image divided into 25x25 tiles, with a library of 1,000 images, tile matching is the most computationally expensive step. Precomputing color averages significantly reduces runtime.

5. Results

Illustrations

1. **Original Image**:

2. **Photo Mosaic** (Tile size: 20x20 pixels):



3. **Zoomed-in Section**:

File Sizes:

- **Original Image**: 1.4 MB
- **Mosaic Image**: 1.4 MB

6. Reflection

Algorithmically:

- The solution is efficient for medium-scale mosaics and produces visually appealing results. Precomputing image features (average color) reduces the computational load during tile matching.

Artistically:

- The mosaic effectively preserves the overall structure of the original image while adding a creative touch. However, finer details may be lost with larger tiles or limited tile libraries